Technical Cybersecurity

Return to Libc

Return to Libc

Stack smashing popular through the 90s

- To protect, engineers implemented Data Execution Prevention (on Windows) or W^X (on Linux)
  - Basically, prohibiting execution on the stack
  - Some sections of memory are executable, and can hold code, others are data, and cannot be executed from

Enter return to libc (or ret2libc)

- Set an address on the stack to redirect execution to a libc function that does something for us (like execute /bin/sh)

Why does this work?

We're not executing anything!

- We write an address to the stack in the RA pointer field
- …address is popped from stack in RET call
- …inserted into EIP/RIP
- …and execution starts in that function

Why does this work?

- Almost every program image has code from libc inserted by the loader
- If we can find it, we can execute it as it's loaded into executable memory

But how do we find it?

GDB!

- …ahh, good old GDB.
- We can use GDB to print the addresses of symbols
- We can get the same information from core files

We can find the address of a function…

- …But that's not enough. It's not like there's a function out there that does exactly what you want. You still need to supply arguments!

Function Arguments

How do arguments get passed to a function?

- Part of a computer's application binary interface (ABI)
- Usually covered in *calling conventions*
  - stdcall, cdecl, etc.

Depends on architecture

- x86: Function args on stack
- x86_64: Function args in registers
- ARM: Some in registers (usually first four), rest on stack
- MIPS: Same as ARM

x86-land ret2libc coming up next!